Working with Relational Databases

Hackathon JDBC

Evaluation Criteria

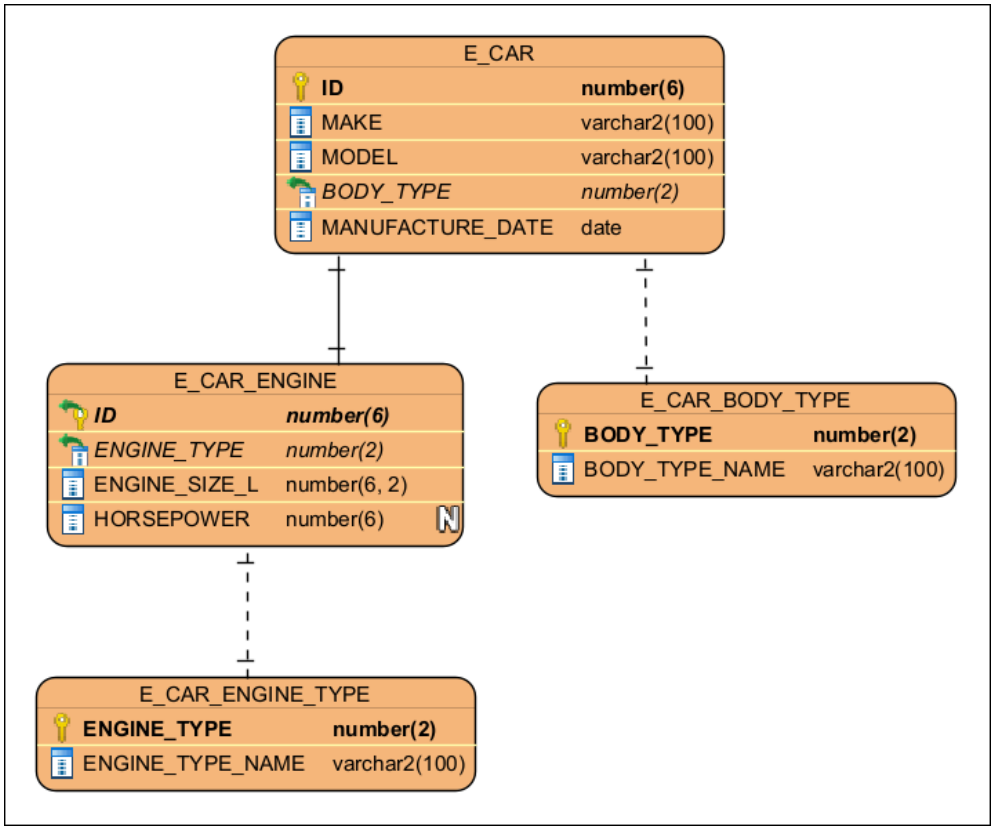
- Does the Java code provide a complete and accurate solution for the stated requirements? Including, but not limited to the following:
    - Do the methods do what they are expected to do?
    - Is the right data returned in the right order?
    - Does the DAO follow the normal practices for that pattern as taught on the course?
- Do the JUnit tests provide adequate coverage of the functionality? Are exceptions used to manage errors that can occur in the code? For example:
    - Are there enough tests to fully cover the functionality?
    - Do the tests follow JUnit and TDD best practices?
    - Are the inserts and deletes performed in a transaction?
    - Are there incorrect results that would pass the tests?
    - Are the tests self-contained and repeatable?
    - Does exception handling follow best practices?
    - Is exception handling tested as far as possible?
- Does the Java code follow best practices? For example:
    - Well-formatted and easy to read
    - Short methods with descriptive names
    - Variables with descriptive names
    - Javadoc comments on new classes and public methods
    - DRY – Don't Repeat Yourself
    - YAGNI – You Ain't Going to Need It
    - SOLID principles
    - PIE
- Does the SQL meet the requirements? Does it follow best practices? For example:
    - Are the correct results returned in the right order?
    - Do the DML statements insert or delete the right rows?
    - Is code well laid out and readable?
    - If tables must be joined, are they joined on the right columns?

Scenario

You have been assigned to a development team that is querying a relational database for information. You have been provided with the following:

1. A database schema and the DDL to create and populate it. You may not change the tables in the schema.
2. Value classes that represent the core data in the schema together with enums that support some of the fields. You may not change the name or data type of any of the fields provided. However, you may create additional methods in these classes.
3. An interface that represents a Data Access Object. You may add methods to the interface if they are appropriate according to object-oriented best practice, but you may not remove any methods since another team has already started working on components that depend on the interface.

Database Schema



Key Relationships:

- E_CAR_ENGINE's primary key ID is also a foreign key that references the primary key of E_CAR.
- E_CAR has a foreign key column BODY_TYPE that references the primary key of E_CAR_BODY_TYPE.
- E_CAR_ENGINE has a foreign key column ENGINE_TYPE that references the primary key of E_CAR_ENGINE_TYPE.

Some points to consider:

- Watch out for optional relationships. In some cases, these may indicate the need for outer joins.
- What do you need to test? Can you test both positive and negative outcomes?

Tasks

- Implement the DAO interface. Note the following additional requirements:
  - The `getCars()` method should retrieve a list of fully populated `Car` objects in ID order. If a car has an engine, its engine should reference a fully populated `CarEngine` object. The returned list should include every car, regardless of whether it has an engine.
  - The `insertCar()` method should insert a car and its engine.
  - The `updateCar()` method should update a car and its engine.
  - The `deleteCar()` method should delete a car and its engine.
- Work TDD. You must provide tests for any functionality that you create. Your tests must be self-contained and repeatable.